July 1991 Report No. STAN-CS-91-1372

AD-A254 368

(1)

A Natural Randomization Strategy for Multicommodity Flow and Related Algorithms

by

DTIC
ELECTE
AUG 2 1 1992
S A D

Andrew V. Goldberg

Department of Computer Science

Stanford University

Stanford, California 94305

This document has been approved for public release and sale; its distribution is unlimited.

92-23194

92 8 19 119

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1991	

4. TITLE AND SUBTITLE

A Natural Randomization Strategy for Multicommodity Flow and Related Algorithms

5. FUNDING NUMBERS

6. AUTHOR(S)

Andrew V. Goldberg

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Science Department
Stanford University
Stanford, CA 94305

8. PERFORMING ORGANIZATION REPORT NUMBER

STAN-CS-91-1372

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

ONR
Arlington, VA 22217

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We consider the approximation algorithm of Leighton et. al. for the multicommodity flow problem. We give a more natural randomization strategy that is simpler than the one in the original algorithm and results in a better running time. This strategy also applies to several related algorithms.

14. SUBJECT TERMS

15. NUMBER OF PAGES

11

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

A Natural Randomization Strategy for Multicommodity Flow and Related Algorithms

*Andrew V. Goldberg**
Department of Computer Science
Stanford University
Stanford, CA 94305

July 1991

DTIC QUALITY INSPECTED 3

Accesion For	
NTIS CRA&I	✓
DTIC TAB	
Unannounced	
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

*Research partially supported by NSF Presidential Young Investigator Grant CCR-8858097 with matching funds from AT&T and DEC, and ONR Young Investigator Award N00014-91-J-1855.

Abstract

We consider the approximation algorithm of Leighton et. al. [8] for the multicommodity flow problem. We give a more natural randomization strategy that is simpler than the one in [8] and results in a better running time. This strategy also applies to several related algorithms.

1 Introduction

The *multicommodity flow* problem is as follows: given a network with capacity constraints and commodity supplies and demands, find a flow that satisfies the demands without violating capacity constraints. The multicommodity flow problem is a classical problem that has numerous applications. The *concurrent flow* problem is an optimization version of the multicommodity flow problem, where the goal is to maximize the fraction of the satisfied demands, *i.e.*, to maximize z such that if the demands are multiplied by z, the resulting multicommodity flow problem is feasible.

In this paper we work with the concurrent flow problem. We denote the number of nodes in the network by n, the number of arcs by m, the number of commodities by k, the largest capacity by U, the largest demand by D, and assume that capacities and demands are integral.

The only known polynomial time algorithms for the problem are based on polynomial-time methods for linear programming, either the ellipsoid method [6] or the interior-point method [5]. The fastest currently known algorithm, due to Vaidya [12], takes advantage of the matrix structure to achieve improvement over the underlying interior-point method. This algorithm runs in $O(k^{3.5}n^3m^{0.5}\log(nDU))$ time.

For many applications it is sufficient to find an approximate solution to the problem, *i.e.*, a feasible solution that ships $(1-\epsilon)z^*$ fraction of the demands, where z^* is the optimal value. In these applications, ϵ is a constant or a slowly decreasing function of n (*e.g.* $\frac{1}{n}$). Shahrokhi and Matula [11] gave an approximation algorithm for a special case of the concurrent flow problem with uniform capacities. Their work motivated several other papers. A faster algorithm for the uniform capacity case was given by Klein et. al. [7]. An algorithm for the general version of the concurrent flow problem was given by Leighton at. al. [8]. The deterministic version of this algorithm runs in $O(k^2(\epsilon^{-2}\log\frac{n}{\epsilon}+\log n\log k))$ minimum-cost flow computations and the randomized version in $O(k(\epsilon^{-3}\log\frac{n}{\epsilon}+\log n\log k))$ minimum-cost flow computations. (For a survey of minimum-cost flow algorithms, see *e.g.* [2].)

In this paper we give a modification of the relaxed optimality conditions that leads to a slightly simpler analysis of the algorithm and allows us to show that the simplest randomization strategy, which selects a commodity to be updated during the next iteration uniformly at random, works better than the more complicated strategy used in [8]. We show an $O(k(\epsilon^{-2}\log\frac{n}{\epsilon} + \log n \log k))$ bound on the number of minimum-cost flow computations.

From the theoretical viewpoint, the dependence on ϵ is a major drawback of the algorithm of [8] since exponential precision is required to solve the multicommodity flow problem exactly. We reduce this dependence by a factor of ϵ^{-1}. Our randomization strategy appears to be a better choice in practice as well.

Our results can also be applied in a straightforward way to improve the results of [7] on the uniform multicommodity flow problem, as well as in the generalizations of the multicommodity flow algorithm to other linear programming problems [10].

A recent paper of Grigoriadis and Khachiyan [4] extends the results of [8] to block-structured convex linear programs. This paper uses a randomization strategy similar to ours in a somewhat different framework.

This paper is organized as follows. Section 2 gives definitions and notation used in the paper. Section 3 describes relaxed optimality conditions. Section 4 discusses the exponential length function and its properties. Section 5 describes the algorithm and Section 6 analyses it. Section 7 discusses some aspects of the algorithm and general applicability of our techniques.

2 Definitions and Notation

In this paper we consider a directed version of the multicommodity flow problem; however, the results also hold for the undirected version with straightforward modifications. An input to the multicommodity flow problem is a graph $G = (V, E)$, a capacity function $u : V \rightarrow R^+$,[1] and a demand specification for commodities. For each commodity $i : 1 \leq i \leq k$, the specification contains the source s_i, the sink t_i, and a nonnegative demand d_i. Without loss of generality we assume that $m \geq n$ and that G has no parallel arcs. We assume that capacities and demands are reals and denote the biggest capacity by U and the biggest demand by D.

[1] R^+ denotes the set of nonnegative reals.

A multicommodity flow f is given by a set of functions f_i, $1 \leq i \leq k$, $f_i : E \rightarrow R^+$. Each function f_i must satisfy *conservation constraints*

$$\forall v \in V, \quad \sum_{(u,v)\in E} f_i(u,v) - \sum_{(v,w)\in E} f_i(v,w) = \begin{cases} d_i & \text{if } v = t_i, \\ -d_i & \text{if } v = s_i, \\ 0 & \text{otherwise.} \end{cases}$$

We define $f(v,w) = \sum_{1\leq i\leq k} f_i(v,w)$. A multicommodity flow is feasible if the *capacity constraints* are satisfied:

$$\forall (v,w) \in E, \quad f(v,w) \leq u(v,w).$$

The *concurrent flow problem* [11] is an optimization version of the multicommodity flow problem where the objective is to maximize z such that the problem with demands zd_i is feasible. An equivalent problem is to minimize λ such that the problem with demands d_i and capacities λu is feasible. Let λ^* denote the optimal value of λ. We define the *congestion* by $\lambda(v,w) = f(v,w)/u(v,w)$, and let $\lambda = \max_{(v,w)\in E} \lambda(v,w)$. A multicommodity flow f is *ϵ-optimal* if $\lambda \leq (1+\epsilon)\lambda^*$. In this paper we consider the problem of finding an ϵ-optimal solution to the problem; throughout the paper we assume that $0 < \epsilon \leq 1$.

Next we introduce the length function, which has its roots in linear programming duality. The *length function* $\ell : E \rightarrow R^+$ is a nonzero, nonnegative function. Let $dist_\ell(v,w)$ denote the distance in G from v to w with respect to ℓ.

To simplify notation, we sometimes view length, capacity, and flow functions as vectors indexed by arcs. Given two vectors a and b, let $a \circ b = \sum_{(v,w)\in E} a(v,w)b(v,w)$.

Theorem 2.1 For a multicommodity flow f and a length function ℓ we have

$$\lambda \ell(v,w) \circ u(v,w) \geq \sum_{i=1}^{k} dist_\ell(s_i,t_i)d_i.$$

Given a length function ℓ, we define the cost of the flow of commodity i by $C_i = f_i \circ \ell$. Let $C_i^*(\lambda)$ be the value of the minimum-cost flow f_i satisfying the demands of commodity i with costs ℓ and capacities λu.

Lemma 2.2 [8] For f, C_i, and $C_i^*(\lambda)$ as above,

$$\lambda \ell \circ u \geq \sum_{i=1}^{k} C_i \geq \sum_{i=1}^{k} C_i^*(\lambda).$$

A multicommodity flow f minimizes λ iff there is a length function ℓ for which the above terms are equal.

Lemma 2.3 [8] For f, ℓ as above,

$$\frac{\sum_{i=1}^{k} C_i^*(\lambda)}{u \circ \ell}$$

is a lower bound on λ^*.

3 Relaxed Optimality

The *relaxed optimality conditions* are as follows:

$$\forall (v,w) \in E \quad (1+\epsilon) f(v,w) \geq \lambda u(v,w) \quad \text{or} \quad u(v,w)\ell(v,w) \leq \frac{\epsilon}{m}(u \circ \ell). \tag{1}$$

$$(1-2\epsilon)\sum_i C_i \leq \sum_i C_i^*(\lambda). \tag{2}$$

The first condition is the same as the one in [8]; it states that either an arc is close to being saturated, or its "volume" is small compared to the total "volume". The second condition states that the total cost of f with respect to ℓ is close to the optimal.

Theorem 3.1 If f, ℓ, and ϵ satisfy the relaxed optimality conditions, then λ is at most $(1 + O(\epsilon))\lambda^*$.

Proof. From condition (1), we have

$$\sum_{(v,w)\in E} ((1+\epsilon) f(v,w)\ell(v,w) + \lambda \frac{\epsilon}{m} u \circ \ell) \geq \lambda u \circ \ell.$$

Rewriting, we get

$$(1+\epsilon) f \circ \ell \geq \lambda u \circ \ell (1-\epsilon)$$

or

$$\frac{1-\epsilon}{1+\epsilon}\lambda u \circ \ell \leq f \circ \ell = \sum_i C_i \leq \frac{1}{1-2\epsilon}\sum C_i^*(\lambda). \tag{3}$$

By Lemma 2.3 and the previous inequality,

$$\lambda^* \geq \frac{\sum_i C_i^*(\lambda)}{u \circ l} \geq \frac{(1-\epsilon)(1-2\epsilon)}{1+\epsilon}\lambda,$$

and the theorem follows. ■

4 Exponential Length Function

Shahrokhi and Matula [11] were the first to use the length function that exponentially depends on flow in the context of the uniform capacities. Leighton et. al. introduced the following generalization of this idea.

Define

$$\ell(v,w) = \frac{e^{\alpha\lambda(v,w)}}{u(v,w)}. \tag{4}$$

Lemma 4.1 [8] Suppose f is defined by (4) and $\alpha \geq (1+\epsilon)\lambda^{-1}\epsilon^{-1}\ln(m\epsilon^{-1})$. Then the first relaxed optimality condition (1) is satisfied.

Proof. Suppose $(1+\epsilon)f(v,w) < \lambda u(v,w)$. Then $\lambda(v,w) < \lambda/(1+\epsilon)$. We need to show that $u(v,w)\ell(v,w) \leq \frac{\epsilon}{m} u \circ \ell$.

Note that

$$\frac{\epsilon}{m}\, u \circ \ell = \frac{\epsilon}{m} \sum_E e^{\alpha\lambda(v,w)} \geq \frac{\epsilon}{m} e^{\alpha\lambda}.$$

To obtain the last inequality, we bounded the sum of nonnegative terms by the largest one.

On the other hand,

$$u(v,w)\ell(v,w) = e^{\alpha\lambda(v,w)} < e^{\alpha\frac{\lambda}{1+\epsilon}} = e^{\alpha\lambda} e^{\frac{-\alpha\lambda\epsilon}{1+\epsilon}}.$$

By the assumption of the lemma, the second term is at most $e^{-\ln(m\epsilon)} = \frac{\epsilon}{m}$, and the lamma follows. ■

5 Algorithm Description

The algorithm maintains a multicommodity flow f and the corresponding exponential length function ℓ such that f satisfies the demands. By Lemma 4.1, the first relaxed optimality

condition is always satisfied. While the second condition is not satisfied, the algorithm picks a random commodity and, if the potential function $u \circ \ell$ decreases, replaces a fraction of its flow by the minimum-cost flow with respect to the cost function ℓ. The expected decrease in the potential function is large, ensuring quick termination of the algorithm with high probability.

Finding initial solution The algorithm starts with an initial solution f such that $\lambda \leq k\lambda^*$. Such a solution is obtained by finding, for each commodity i, a maximum flow g_i in the network with source s_i, sink t_i, and capacity function u, and setting $f_i(v,w) = g_i(v,w)\frac{d_i}{|g_i|}$, where $|g_i|$ denotes the value of g_i. It is easy to see that for the resulting flow f, $\lambda \leq k\lambda^*$. The length function ℓ defined by the initial flow is also computed.

Improving the current solution At each iteration, the algorithm iteratively improves the current flow, until the desired approximation precision is obtained. An iteration (*rerouting step*) works as follows.

1. Select commodity i from the set $\{1,\ldots,k\}$ uniformly at random.
2. Consider the network with capacity function λu and cost function ℓ. Compute the minimum cost flow f_i^* in this network satisfying demands for the commodity i.
3. Define $f_i' = (1-\sigma)f_i + \sigma f_i^*$, and let f' be the multicommodity flow obtained from f by replacing f_i by f_i'.
4. Compute the length function ℓ' for f'.
5. If $u \circ \ell > u \circ \ell'$ then replace f by f'.

The parameters α and σ are set as follows: $\alpha = 2(1+\epsilon)\lambda^{-1}\epsilon^{-1}\ln(m\epsilon^{-1})$ and $\sigma = \frac{\epsilon}{4\alpha\lambda}$. The values of these parameters are *not* updated at every iteration, but the values are updated when λ decreases by at least a factor of two from its value during the time of the last update. Intuitively, an iteration of the algorithm replaces a σ fraction of a random commodity by the same fraction of the optimal flow of this commodity. The flow is updated only if the potential function $u \circ \ell$ decreases.

Nonscaling and scaling algorithms A simple implementation of the algorithm is to set ϵ to the desired value at the very beginning, and improve the flow until it becomes ϵ-optimal.

An alternative is to scale ϵ. Note, however, that we assumed that $\epsilon \leq 1$. Thus the scaling algorithm starts with a constant ϵ (say 1) and finds an ϵ-optimal solution $f^{(0)}$. Then at jth scaling iteration, ϵ is reduced by a factor of two and a new ϵ-optimal solution $f^{(j)}$ is computed starting from $f^{(j-1)}$, until the desired precision is reached.

As we shall see later, the use of scaling improves the running time of the algorithm by a factor of ϵ^{-1} if ϵ is small enough.

Termination detection The above description assumes that we know when the current flow becomes ϵ-optimal. This is not the case; however, we can test for ϵ-optimality by computing minimum-cost flows of all commodities. This can be done every k iterations while increasing the worst-case running time bound by a constant factor.

6 Algorithm Analysis

For the purpose of the analysis, we define the potential function Φ by $\Phi = u \circ \ell$.

The next lemma shows that when f_i is rerouted, Φ decreases by almost $\alpha\sigma(C_i - C_i^*(\lambda))$. The proof of this lemma is similar to the proof of [8] showing that Φ decreases significantly if a "bad" commodity is rerouted.

Let Φ and Φ_a be the values of the potential function before and after the rerouting, respectively.

Lemma 6.1

$$\Phi - \Phi_a \geq \alpha\sigma(C_i - C_i^*(\lambda) - \epsilon \mathbf{C}_i).$$

Proof. Let ℓ and ℓ_a be the length functions before and after the rerouting. By Taylor's theorem, for $|t| \leq \epsilon/4 \leq 1/4$ we have

$$e^{x+t} \leq e^x + te^x + \frac{\epsilon}{2}|t|e^x.$$

Thus

$$\ell_a(v,w) \leq \ell(v,w) + \frac{\alpha\sigma}{u(v,w)}(f_i^*(v,w) - f_i(v,w))\ell(v,w) + \frac{\epsilon\alpha\sigma}{2u(v,w)}|f_i^*(v,w) - f_i(v,w)|\ell(v,w)$$

(note that $f_i^*(v,w) - f_i(v,w) \leq \lambda u(v,w)$, and recall the choice of σ). We have

$$\begin{aligned}
\Phi - \Phi_a &= (\ell - \ell_a) \circ u \\
&\geq \alpha\sigma(f_i - f_i^*) \circ \ell - \frac{\epsilon\alpha\sigma}{2}|f_i^* - f_i| \circ \ell \\
&\geq \alpha\sigma(C_i - C_i^*(\lambda)) - \epsilon\alpha\sigma C_i.
\end{aligned}$$

The last line follows by definition of C_i and $C_i^*(\lambda)$ and by the fact that $C_i \geq C_i^*(\lambda) \geq 0$. ■

Note that if the current flow does not satisfy the second relaxed optimality condition, then the expected value of $C_i - C_i^*(\lambda)$ is large and Φ decreases significantly. The following lemma formalizes this statement.

Lemma 6.2 Suppose $\frac{\epsilon}{8\alpha\lambda} \leq \sigma \leq \frac{\epsilon}{4\alpha\lambda}$ and f does not satisfy the second relaxed optimality condition. Then the expected decrease in Φ due to a rerouting step is $\Omega(\frac{\epsilon^2}{k})$.

Proof. Since the algorithm selects a commodity uniformly at random, we have

$$\begin{aligned}
E[\Phi - \Phi_a] &\geq \frac{1}{k}\alpha\sigma\sum_i(C_i - C_i^*(\lambda) - \epsilon C_i) \\
&= \frac{\alpha\sigma}{k}(\sum_i((1-2\epsilon)C_i - C_i^*) + \epsilon\sum_i C_i) \\
&\geq \frac{\alpha\sigma}{k}\frac{1-\epsilon}{1+\epsilon}\epsilon\lambda\Phi \\
&= \Omega(\frac{\epsilon^2}{k}).
\end{aligned}$$

The third line follows from the assumption that f does not satisfy the second relaxed optimality condition and since by (3)

$$\frac{1-\epsilon}{1+\epsilon}\lambda\Phi \leq \sum_i C_i.$$

The last line follows from the assumption on σ. ■

Note that since α and σ are updated every time λ decreases by a factor of two, conditions of the lemmas 4.1 and 6.2 are always satisfied.

Next we analyze the nonscaling algorithm.

Lemma 6.3 Suppose the parameters α and σ are set when the congestion is λ_0. In expected $O(\epsilon^{-3}k\log\frac{n}{\epsilon})$ iterations, either the algorithm terminates or λ decreases to $\lambda_0/2$ or less.

Proof. Initially, $\Phi \leq me^{\alpha\lambda_0}$ since the congestion on any arc is at most λ_0. Note that until λ decreases by a factor of two, the congestion on some arc is at least $\lambda_0/2$, so $\Phi \geq e^{\alpha\lambda_0/2}$. Thus Φ cannot decrease by more than a factor of $me^{\alpha\lambda_0/2}$ without λ getting below $\lambda_0/2$.

By Lemma 6.2, $O(\epsilon^{-2}k)$ iterations reduce Φ by a factor of two. Φ can be halved at most $O(\log(me^{\alpha\lambda_0/2})) = O(\epsilon^{-1}\log\frac{n}{\epsilon})$ times before λ is halved. Therefore in $O(\epsilon^{-3}k\log\frac{n}{\epsilon})$ iterations (expected), the algorithm terminates or λ is halved. ■

Note that the work done by an iteration of the algorithm is dominated by a minimum-cost flow computation. Combined with the above lemma and the fact that initially $\lambda \leq k\lambda^*$, this yields the following result.

Theorem 6.4 The nonscaling algorithm runs in expected $O(\epsilon^{-3}k\log k\log\frac{n}{\epsilon}))$ minimum-cost flow computations.

To analyze the scaling algorithm, we need the following version of Lemma 6.3.

Lemma 6.5 Suppose the parameters α and σ are set when the congestion is λ_0, and the initial flow is $O(\epsilon)$-optimal. Then in expected $O(\epsilon^{-2}k\log\frac{n}{\epsilon})$ iterations, either the algorithm terminates or λ decreases by at least a factor of two.

Proof. Recall that $\epsilon \leq 1$, so $\lambda_0 = O(\lambda^*)$. The proof is similar to that of Lemma 6.3, except that the upper bound on Φ is $me^{\alpha(1+O(\epsilon)\lambda^*)}$ and the lower bound is $e^{\alpha\lambda^*}$, so Φ needs to decrease by a factor of $me^{\alpha O(\epsilon)\lambda_0}$. ■

The scaling algorithm starts with $\epsilon = O(1)$, and obtains an $O(1)$-optimal solution in $O(k\log k\log m)$ iterations by Theorem 6.4. Then the scaling process starts. Since the number of rerouting steps needed to reduce ϵ by a factor of two is proportional to ϵ^{-2}, the last scaling iteration dominates. This iteration terminates in $O(\epsilon^{-2}k\log\frac{n}{\epsilon})$ rerouting steps. We have

Theorem 6.6 The scaling algorithm runs in expected $O(k(\epsilon^{-2}\log\frac{n}{\epsilon} + \log k\log n))$ minimum-cost flow computations.

7 Remarks

First, we would like to address the selection of a minimum-cost flow subroutine. Since the costs in the minimum-cost subproblems can be big due to the use of the exponential length function, Orlin's strongly polynomial algorithm [9] appears to be the best choice. However, it is enough to solve the subproblems approximately, so the costs can be rounded to small integers. When this is done, the cost scaling algorithm of Goldberg and Tarjan [3] or the double scaling algorithm of Ahuja et. al. [1] become a better choice. The selection of the subroutine is discussed in more detail in [8].

The deterministic version of the algorithm of Leighton et. al. [8] finds a commodity i with the biggest $C_i - C_i^*(\lambda)$ and reroutes this commodity. Our variant of the relaxed optimality conditions also can be used to analyze this algorithm. Since the maximum of a set of numbers is at least as big as the average, the deterministic choice of i gives an improvement in Φ that is at least as big as the expected improvement. Thus the bound on the number of rerouting steps apply for the deterministic version of the algorithm. Each deterministic rerouting step, however, requires k minimum-cost flow computations (to compute $C_j^*(\lambda)$ for every commodity j); in contrast, a randomized rerouting step requires a single minimum-cost flow computation.

Our randomization strategy can also be applied to the algorithm of Klein et. al. [7]. This improves the running time bound of the randomized algorithm of [7] by a factor of ϵ^{-1}. We omit the details, which are straightforward given those of [7], [8], and this paper. A recent paper of Plotkin, Shmoys, and Tardos [10], that extends the results of [8] to a more general class of linear programming problems, also extends our randomization strategy to the more general framework.

Acknowledgements

The author would like to thank Éva Tardos and Serge Plotkin for helpful discussions, and Robert Kennedy and Tomasz Radzik for comments on a draft of this paper.

References

[1] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. Finding Minimum-Cost Flows by Double Scaling. Technical Report STAN-CS-88-1227, Department of

Computer Science, Stanford University, 1988.

[2] A. V. Goldberg, É. Tardos, and R. E. Tarjan. Network Flow Algorithms. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Flows, Paths, and VLSI Layout*, pages 101–164. Springer Verlag, 1990.

[3] A. V. Goldberg and R. E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Math. of Oper. Res.*, 15:430–466, 1990. A preliminary version appeared in *Proc. 19th ACM Symp. on Theory of Comp.*, 7-18, 1987.

[4] M. D. Grigoriadis and L. G. Khachiyan. Fast Approximation Schemes for Convex Programs with Many Blocks and Coupling Constraints. Technical Report DCS-TR-273, Department of Computer Science, Rutgers University, 1991.

[5] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4:373–395, 1984.

[6] L. G. Khachian. Polynomial Algorithms in Linear Programming. *Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki*, 20:53–72, 1980.

[7] P. Klein, S. A. Plotkin, C. Stein, and É. Tardos. Faster Approximation Algorithms for the Unit Capacity Concurrent Flow Problem with Applications to Routings and Finding Sparse Cuts. Technical Report 961, School of ORIE, Cornell University, 1991.

[8] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast Approximation Algorithms for Multicommodity Flow Problems. In *Proc. 23st Annual ACM Symposium on Theory of Computing*, 1991.

[9] J. B. Orlin. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 377–387, 1988.

[10] S. A. Plotkin, D. Shmoys, and É. Tardos. Fast Approximation Algorithms for Fractional Packing and Covering. In *Proc. 32nd IEEE Annual Symposium on Foundations of Computer Science*, 1991. To appear.

[11] F. Shahrokhi and D. Matula. The Maximum Concurrent Flow Problem. *J. Assoc. Comput. Mach.*, 37:318–334, 1990.

[12] P. M. Vaidya. Speeding up Linear Programming Using Fast Matrix Multiplication. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989.